(2)

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
**Ada Compiler Validation Summary Report:** New York University NYU Ada/Ed, Version 1.10, VAX 8600 (Host & Target), 890523W1.10086

5. TYPE OF REPORT & PERIOD COVERED
22 May 1989 to 22 May 1990

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Wright-Patterson AFB
Dayton, OH, USA

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
Wright-Patterson AFB
Dayton, OH, USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Wright-Patterson AFB
Dayton, OH, USA

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
UNCLASSIFIED



18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)
Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
New York University, NYU Ada/Ed, Version 1.10, Wright-Patterson AFB, VAX 8600 under VMS, Version 4.7 (Host & Target), ACVC 1.10.



DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

AVF Control Number: AVF-VSR-299.0689
89-01-24-NYU

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890523W1.10086
New York University
NYU Ada/Ed, Version 1.10
VAX 8600

Completion of On-Site Testing:
22 May 1989

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: NYU Ada/Ed, Version 1.10

Certificate Number: 890523W1.10086

Host: VAX 8600 under
VMS, Version 4.7

Target: VAX 8600 under
VMS, Version 4.7

Testing Completed 22 May 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Steven P. Wilson

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

John F. Kramer

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

for

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

## 1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 22 May 1989 at New York NY.

## 1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

## 1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

## 1.4 DEFINITION OF TERMS

| | |
|---|---|
| ACVC | The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language. |
| Ada Commentary | An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd. |
| Ada Standard | ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987. |
| Applicant | The agency requesting validation. |
| AVF | The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines. |
| AVO | The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices. |
| Compiler | A processor for the Ada language. In the context of this report, a compiler is any language processor, including |

cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — An ACVC test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

## 1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every

illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate

tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

# CHAPTER 2

# CONFIGURATION INFORMATION

## 2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: NYU Ada/Ed, Version 1.10

ACVC Version: 1.10

Certificate Number: 890523W1.10086

Host Computer:

| | |
|---|---|
| Machine: | VAX 8600 |
| Operating System: | VMS<br>Version 4.7 |
| Memory Size: | 28 Megabytes |

Target Computer:

| | |
|---|---|
| Machine: | VAX 8600 |
| Operating System: | VMS<br>Version 4.7 |
| Memory Size: | 28 Megabytes |

## 2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to six levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined type LONG_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) Some of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses some extra bits for extra range. (See test C35903A.)

(4) Sometimes CONSTRAINT_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..Z.)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z.)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z.)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

(2) No exception is raised when 'LENGTH is applied to a null array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) No exception is raised when 'LENGTH is applied to a null array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises STORAGE_ERROR when the array objects are declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises STORAGE_ERROR when the array objects are declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B, EA3004C..D, and CA3004E..F.)

i. Generics

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output

(1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J, CE2102R, CE2102T, and CE2102V.)

(5) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K.)

(6) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(7) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(8) RESET and DELETE operations are supported for text files. (See tests CE3102F..G, CE3104C, CE3110A, and CE3114A.)

(9) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(10) Temporary sequential files are given names and not deleted when closed. (See test CE2108A.)

(11) Temporary direct files are given names and not deleted when closed. (See test CE2108C.)

(12) Temporary text files are given names and not deleted when closed. (See test CE3112A.)

(13) Only one internal file can be associated with each external file for sequential files. (See tests CE2107A..E, CE2102L, CE2110B, and CE2111D.)

(14) Only one internal file can be associated with each external file for direct files. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(15) Only one internal file can be associated with each external file for text files. (See tests CE3111A..E, CE3114B, and CE3115A.)

# CHAPTER 3

# TEST INFORMATION

## 3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 194 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. Modifications to the code, processing, or grading for 15 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

## 3.2 SUMMARY OF TEST RESULTS BY CLASS

| RESULT | TEST CLASS | | | | | | TOTAL |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | L | |
| Passed | 129 | 1128 | 2137 | 15 | 26 | 44 | 3479 |
| Inapplicable | 0 | 10 | 178 | 2 | 2 | 2 | 194 |
| Withdrawn | 1 | 2 | 35 | 0 | 6 | 0 | 44 |
| TOTAL | 130 | 1140 | 2350 | 17 | 34 | 46 | 3717 |

## 3.3 SUMMARY OF TEST RESULTS BY CHAPTER

| RESULT | CHAPTER | | | | | | | | | | | | | TOTAL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |
| Passed | 194 | 648 | 645 | 242 | 170 | 99 | 160 | 331 | 131 | 36 | 252 | 290 | 281 | 3479 |
| Inappl | 18 | 1 | 35 | 6 | 2 | 0 | 6 | 1 | 6 | 0 | 0 | 79 | 40 | 194 |
| Wdrn | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 35 | 4 | 44 |
| TOTAL | 213 | 650 | 680 | 248 | 172 | 99 | 166 | 334 | 137 | 36 | 253 | 404 | 325 | 3717 |

## 3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

| | | | | | |
|---|---|---|---|---|---|
| E28005C | A39005G | B97102E | C97116A | BC3009B | CD2A62D |
| CD2A63A | CD2A63B | CD2A63C | CD2A63D | CD2A66A | CD2A66B |
| CD2A66C | CD2A66D | CD2A73A | CD2A73B | CD2A73C | CD2A73D |
| CD2A76A | CD2A76B | CD2A76C | CD2A76D | CD2A81G | CD2A83G |
| CD2A84M | CD2A84N | CD2B15C | CD2D11B | CD5007B | CD50110 |
| ED7004B | ED7005C | ED7005D | ED7006C | ED7006D | CD7105A |
| CD7203B | CD7204B | CD7205C | CD7205D | CE2107I | CE3111C |
| CE3301A | CE3411B | | | | |

See Appendix D for the reason that each of these tests was withdrawn.

## 3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 194 tests were inapplicable for the reasons indicated:

a. C24113H..Y (18 tests) are not applicable because the source line exceeds the maximum of 120 characters.

b. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

c. The following 16 tests are not applicable because this implementation does not support a predefined type SHORT_INTEGER:

| | | | | |
|---|---|---|---|---|
| C45231B | C45304B | C45502B | C45503B | C45504B |
| C45504E | C45611B | C45613B | C45614B | C45631B |
| C45632B | B52004E | C55B07B | B55B09D | B86001V |
| CD7101E | | | | |

d. The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

| | | | | |
|---|---|---|---|---|
| C45231C | C45304C | C45502C | C45503C | C45504C |
| C45504F | C45611C | C45613C | C45614C | C45631C |
| C45632C | B52004D | C55B07A | B55B09C | B86001W |
| CD7101F | | | | |

e. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

f. C45531K..L (2 tests) and C45532K..L (2 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 32.

g. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 47.

h. D64005F and D64005G are not applicable because this implementation does not support nesting 10 levels of recursive procedure calls.

i. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

j. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

k. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

l. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not applicable because this implementation does not support pragma INLINE.

m. The following 76 tests are not applicable because, for this implementation, address clause is not supported.

| | | | |
|---|---|---|---|
| CD5003B..I | CD5011A..I | CD5011K..N | CD5011Q..S |
| CD5012A..J | CD5012L..M | CD5013A..I | CD5013K..O |
| CD5013R..S | CD5014A..O | CD5014R..Z | |

n. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

o. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

p. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

q. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

r. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

s. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

t. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

u. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

v. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

w. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

x. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

y. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

z. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

aa. CE2102V is inapplicable because this implementation supports open with OUT_FILE mode for DIRECT_IO.

ab. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ac. CE2107A..E (5 tests), CE2107L, CE2110B, and CE2111D are not applicable because multiple internal files cannot be associated with the same external file for sequential files. The proper exception is raised when multiple access is attempted.

ad. CE2107F..H (3 tests), CE2110D, and CE2111H are not applicable because multiple internal files cannot be associated with the same external file for direct files. The proper exception is raised when multiple access is attempted.

ae. CE3102E is inapplicable because this implementation supports

CREATE with IN_FILE mode for text files.

af. CE3102F is inapplicable because this implementation supports RESET for text files.

ag. CE3102G is inapplicable because this implementation supports deletion of an external file for text files.

ah. CE3102I is inapplicable because this implementation supports CREATE with OUT_FILE mode for text files.

ai. CE3102J is inapplicable because this implementation supports OPEN with IN_FILE mode for text files.

aj. CE3102K is inapplicable because this implementation supports OPEN with OUT_FILE mode for text files.

ak. CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file for text files. The proper exception is raised when multiple access is attempted.

## 3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 15 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B49003A B49005A B84004A B91003A B91003C BD5005B
BE3001A BE3002A

The following modifications were made to compensate for legitimate implementation behavior.

C45651A was modified because membership test uses an upper bound that may be greater than DECIMAL_M4'BASE'LAST. On line 256, the value 1024.0 was replaced by 979.0.

CC3126A was modified because an uninitialized string variable raises

PROGRAM_ERROR. The initializing expression ":= (OTHERS => 'H')" was inserted into variable H's declaration on line 117.

CD2A31A, CD2A32A, and CD2A32E (3 tests) were modified to reduce the number of iterations in each of four "for loops" from 201 to 2 so as to compensate for the SETL implementation's extraordinary consumption of space and time in the execution of these tests. The AVO approved these modifications in preference to effectively forcing the implementation to be changed so that SIZE length clauses were not supported at all. Other tests in these two series contain a single, similar loop; they were successfully processed in approximately an hour each.

CD2C11A..B (2 tests) were modified because this implementation raises PROGRAM_ERROR when procedure TEST_TASK is called with an uninitialized actual parameter, W. The initializing expression ":= 5.0" was inserted into variable W's declaration on line 41 and 44 respectively.

## 3.7 ADDITIONAL TESTING INFORMATION

### 3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the NYU Ada/Ed was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

### 3.7.2 Test Method

Testing of the NYU Ada/Ed using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

| | |
|---|---|
| Host computer: | VAX 8600 |
| Host operating system: | VMS, Version 4.7 |
| Target computer: | VAX 8600 |
| Target operating system: | VMS, Version 4.7 |
| Compiler: | NYU Ada/Ed, Version 1.10 |

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the VAX 8600. Results were transferred to a workstation and printed directly onto the host computer.

The compiler was tested using command scripts provided by New York University and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

| OPTION | EFFECT |
|---|---|
| A | Specifies the source file. |
| L | Specifies the listing file. |
| LIBFILE | Specifies the program library. |
| NEWLIB | Specifies that a new program library is to be used. |
| AISFILE | Specifies the name of the intermediate files. |
| MAIN | Specifies the name of the main program unit if there can be more than one. |
| MEMORY_SIZE | Specifies the maximum memory to be used. |

Tests were compiled, linked, and executed (as appropriate) using 3 computers. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

### 3.7.3 Test Site

Testing was conducted at New York NY and was completed on 22 May 1989.

# APPENDIX A

## DECLARATION OF CONFORMANCE

New York University has submitted the following Declaration of Conformance concerning the NYU Ada/Ed Compiler.

# DECLARATION OF CONFORMANCE

Compiler Implementor: New York University
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.10

## Base Configuration

| | | | |
|---|---|---|---|
| Base Compiler Name: | NYU Ada/Ed | Version: | 1.10 |
| Host Architecture ISA: | VAX-8600 | OS&VER: | VMS version 4.7 |
| Target Architecture ISA: | VAX-8600 | OS&VER: | VMS version 4.7 |

## Implementor's Declaration

I, the undersigned, representing New York University, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that New York Univesity is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compilers listed in this declaration shall be made only in owner's corporate name.

Bernard Banner Date: 5/22/89

New York University
Bernard Banner, Asst Research Scientist

## Owner's Declaration

I, the undersigned, representing New York University, take full responsibilty for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Bernard Banner Date: 5/22/89

New York University
Bernard Banner, Asst Research Scientist

# APPENDIX B

## APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the NYU Ada/Ed, Version 1.10, as described in this Appendix, are provided by New York University. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...

   type INTEGER is range -1073741823 .. 1073741823;

   type FLOAT is digits 6 range -1.70141E+38 .. 1.70141E+38;
   type LONG_FLOAT is digits 33
        range -6.70390396497129854978701249910292E+153 ..
              6.70390396497129854978701249910292E+153;

   type DURATION is delta 0.01 range -86400.00 .. 86400.00;

   ...

end STANDARD;
```

APPENDIX F OF THE Ada STANDARD

Appendix F: Implementation Dependent Characteristics

(1) The form, allowed places, and effect of implementation dependent pragmas.

NYU Ada/Ed does not recognize any implementation dependent pragmas. The language defined pragmas are correctly recognized and their legality is checked, but, with the exception of LIST and PRIORITY, they have no effect on the execution of the program. A warning message is generated to indicate that the pragma is ignored by NYU ADA/Ed.

(2) The name and the type of every implementation dependent attribute.

There are no implementation dependent attributes in NYU Ada/Ed.

(3) The specification of the package system.

```
package SYSTEM is
   type NAME        is (ADA_ED);
   type ADDRESS     is new INTEGER;
   SYSTEM_NAME      : constant  NAME  :=  ADA_ED;
   STORAGE_UNIT     : constant  :=  32;
   MEMORY_SIZE      : constant  :=  2**30 - 1;
   -- System Dependent Named Numbers:
   MIN_INT          : constant  :=  -(2**30 - 1);
   MAX_INT          : constant  :=  2**30 - 1;
   MAX_DIGITS       : constant  :=  33;
   MAX_MANTISSA     : constant  :=  31;
   FINE_DELTA       : constant  :=  2.0 ** (-31);
   TICK             : constant  :=  0.01;
   -- Other System Dependent Declarations
   subtype PRIORITY is INTEGER range 0 .. 9;
   SYSTEM_ERROR  :  exception;
                    -- raised if internal check fails
end SYSTEM;
```

(4) The list of all restrictions on representation clauses.

NYU Ada/Ed does not support any address clauses.

(5) The conventions used for system generated names.

NYU Ada/Ed does not provide any system generated names denoting system dependent entities, since in any case, representation specifications are not permitted.

(6) The interpretation of expressions that appear in address clauses.

Address expressions in NYU/AdaEd are meaningless, since the model used for interpretation does not use addresses. The ADDRESS type defined in SYSTEM is present only for completeness, and to be able to recognize semantically legal uses of the attribute ADDRESS.

(7) Restrictions on unchecked conversion.

NYU Ada/Ed will correctly recognize and check the validity of any use of unchecked conversion. However, any program which executes an unchecked conversion is considered to be erroneous, and the exception PROGRAM_ERROR will be raised.

(8) Implementation dependent characteristics of the input-output package.

A) Temporary files are fully supported. The naming convention used is as follows:

```
XHHMMSS.TMP
 X stands for the file accessing method
   S - SEQUENTIAL_IO
   D - DIRECT_IO
   T - TEXT_IO
 HH - hour of file creation
 MM - minute of file creation
 SS - second of file creation
```

B) Deletion of files is fully supported.

C) Only one internal file may be associated with the same external file (No multiple accessing of files allowed).

D) File names used in the CREATE and OPEN procedures are standard VMS file names. The function FORM returns the string given as FORM parameter when a file is created. No system-dependent characteristics are associated with that parameter.

E) A maximum of 17 files can be open at any given time during program execution.

F) The standard default input file may be specified using the DATA parameter of the ADA commands. If a file is specified it must be possible to open it at the beginning of program execution, otherwise the exception PROGRAM_ERROR will be raised. If no file is specified SYS$INPUT will be used. The standard output file is SYS$OUTPUT.

G) SEQUENTIAL_IO and DIRECT_IO support constrained array types, record types without discriminants and record types with discriminants with defaults.
H) I/O on access types is possible, but usage of access values read in another program execution is erroneous.

I) Normal termination of the main program causes all open files to be closed, and all temporary files to be deleted.

J) LOW_LEVEL_IO is not supported.

K) The form feed character (CTRL L - ascii 12) is used as the page terminator indicator. Its use as a data element of a file is therefore undefined.

## APPENDIX C

## TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

| Name and Meaning | Value |
|---|---|
| $ACC_SIZE<br>An integer literal whose value is the number of bits sufficient to hold any value of an access type. | 32 |
| $BIG_ID1<br>An identifier the size of the maximum input line length which is identical to $BIG_ID2 except for the last character. | (1..119 => 'A', 120 => '1') |
| $BIG_ID2<br>An identifier the size of the maximum input line length which is identical to $BIG_ID1 except for the last character. | (1..119 => 'A', 120 => '2') |
| $BIG_ID3<br>An identifier the size of the maximum input line length which is identical to $BIG_ID4 except for a character near the middle. | (1..59 => 'A', 60 => '3',<br>61..120 => 'A') |

TEST PARAMETERS

| Name and Meaning | Value |
|---|---|
| $BIG_ID4<br>An identifier the size of the maximum input line length which is identical to $BIG_ID3 except for a character near the middle. | (1..59 => 'A', 60 => '4',<br>61..120 => 'A') |
| $BIG_INT_LIT<br>An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length. | (1..117 => '0',<br>118..120 => "298") |
| $BIG_REAL_LIT<br>A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length. | (1..115 => '0',<br>116..120 => "690.0") |
| $BIG_STRING1<br>A string literal which when catenated with $BIG_STRING2 yields the image of $BIG_ID1. | (1 => '"', 2..61 => 'A',<br>62 => '"') |
| $BIG_STRING2<br>A string literal which when catenated to the end of $BIG_STRING1 yields the image of $BIG_ID1. | (1 => '"', 2..60 => 'A',<br>61..62 => "1"") |
| $BLANKS<br>A sequence of blanks twenty characters less than the size of the maximum line length. | (1..100 => ' ') |
| $COUNT_LAST<br>A universal integer literal whose value is TEXT_IO.COUNT'LAST. | 32_767 |
| $DEFAULT_MEM_SIZE<br>An integer literal whose value is SYSTEM.MEMORY_SIZE. | 1_073_741_823 |
| $DEFAULT_STOR_UNIT<br>An integer literal whose value is SYSTEM.STORAGE_UNIT. | 32 |

| Name and Meaning | Value |
|---|---|
| $DEFAULT_SYS_NAME<br>The value of the constant SYSTEM.SYSTEM_NAME. | ADA_ED |
| $DELTA_DOC<br>A real literal whose value is SYSTEM.FINE_DELTA. | (1..18 => "0.000_000_000_465_",<br>19..35 => "661_287_307_739_2",<br>36..43 => "57_812_5") |
| $FIELD_LAST<br>A universal integer literal whose value is TEXT_IO.FIELD'LAST. | 100 |
| $FIXED_NAME<br>The name of a predefined fixed-point type other than DURATION. | NO_SUCH_TYPE_AVAILABLE |
| $FLOAT_NAME<br>The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT. | NO_SUCH_TYPE_AVAILABLE |
| $GREATER_THAN_DURATION<br>A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION. | 75_000.0 |
| $GREATER_THAN_DURATION_BASE_LAST<br>A universal real literal that is greater than DURATION'BASE'LAST. | 131_073.0 |
| $HIGH_PRIORITY<br>An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY. | 9 |
| $ILLEGAL_EXTERNAL_FILE_NAME1<br>An external file name which is too long. | (1..256 => 'A') |
| $ILLEGAL_EXTERNAL_FILE_NAME2<br>An external file name which contains invalid characters. | "/junk/junk" |
| $INTEGER_FIRST<br>A universal integer literal whose value is INTEGER'FIRST. | -1_073_741_823 |

TEST PARAMETERS

| Name and Meaning | Value |
|---|---|
| $INTEGER_LAST<br>A universal integer literal whose value is INTEGER'LAST. | 1_073_741_823 |
| $INTEGER_LAST_PLUS_1<br>A universal integer literal whose value is INTEGER'LAST + 1. | 1_073_741_824 |
| $LESS_THAN_DURATION<br>A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION. | -75_000.0 |
| $LESS_THAN_DURATION_BASE_FIRST<br>A universal real literal that is less than DURATION'BASE'FIRST. | -131_073.0 |
| $LOW_PRIORITY<br>An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY. | 0 |
| $MANTISSA_DOC<br>An integer literal whose value is SYSTEM.MAX_MANTISSA. | 31 |
| $MAX_DIGITS<br>Maximum digits supported for floating-point types. | 33 |
| $MAX_IN_LEN<br>Maximum input line length permitted by the implementation. | 120 |
| $MAX_INT<br>A universal integer literal whose value is SYSTEM.MAX_INT. | 1073741823 |
| $MAX_INT_PLUS_1<br>A universal integer literal whose value is SYSTEM.MAX_INT+1. | 1_073_741_824 |
| $MAX_LEN_INT_BASED_LITERAL<br>A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be $MAX_IN_LEN long. | (1..2 => "2:", 3..117 => '0',<br>118..120 => "11:") |

| Name and Meaning | Value |
|---|---|
| $MAX_LEN_REAL_BASED_LITERAL<br>A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long. | (1..3 => "16:", 4..116 => '0',<br>117..120 => "F.E:") |
| $MAX_STRING_LITERAL<br>A string literal of size $MAX_IN_LEN, including the quote characters. | (1 => '"', 2..119 => 'A',<br>120 => '"') |
| $MIN_INT<br>A universal integer literal whose value is SYSTEM.MIN_INT. | -1073741823 |
| $MIN_TASK_SIZE<br>An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body. | 128 |
| $NAME<br>A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER. | NO_SUCH_TYPE_AVAILABLE |
| $NAME_LIST<br>A list of enumeration literals in the type SYSTEM.NAME, separated by commas. | ADA_ED |
| $NEG_BASED_INT<br>A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT. | 16#FFFFFFFE# |
| $NEW_MEM_SIZE<br>An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE. | 1_073_741_823 |

TEST PARAMETERS

| Name and Meaning | Value |
|---|---|
| $NEW_STOR_UNIT<br>An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT. | 32 |
| $NEW_SYS_NAME<br>A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value. | ADA_ED |
| $TASK_SIZE<br>An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter. | 128 |
| $TICK<br>A real literal whose value is SYSTEM.TICK. | 0.01 |

## APPENDIX D

## WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this text that must appear at the top of the page.

b. A39005G: This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

c. B97102E: This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

d. C97116A: This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

e. BC3009B: This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

f. CD2A62D: This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

g. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These

tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

h. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86, 96, and 58, respectively).

i. CD2B15C and CD7205C: These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

j. CD2D11B: This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

k. CD5007B: This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

l. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

m. CD7105A: This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

n. CD7203B and CD7204B: These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

o. CD7205D: This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

p. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).